[0076] **FIG. 8** also depicts the modified web service WS1 API contract **754** as seen by the client application **15** with method calls, i.e., client calls **701**, that contain an additional parameter "gparam1" of type "AuthID". For example, this additional parameter may be an identifier that the client application **15** is authorized to use the method in the web service **25**. In this example, the additional parameter is placed as the first parameter to all client calls **701**. Other parameters may be added to the client calls **701**. Thus, the three corresponding examples of client calls **701** in the client application **15** are:

[0077] method1(AuthID gparam1, string param1, int param2, MyType param3);

[0078] method2(AuthID gparam1, char param1, MyType param2); and

[0079] method3(AuthID gparam1).

[0080] Additionally, if it is a requirement for a web service **25** to have certain parameter types converted to other types (from the perspective of the client caller), these parameters, along with the associated target types, and the conversion method would be described in the web service registry **530**, or an additional table indexed off of the web service registry **530**.

[0081] By modifying the API contract, the gateway module **500** can make it appear that any web service **25** has any set of methods as desired, regardless of what methods the service actually implements. As long as calls to these methods are honored somewhere inside of the web services infrastructure **501**, it appears to the client application **15** as though the client method was really implemented by the web service **25**. The web service **25** remains unaware that the client method exists let alone that the separate client call **701** was made. In general the web services infrastructure **501** is able to dynamically modify the methods that would appear to be offered by a web service **25**; even adding methods that might have a central implementation somewhere inside of the web services infrastructure. For example, for more convenient calling on the client side, the web service provider **20** may add a DoMethod( ) method that takes an enumeration of the methods offered by a given web service **25** to each web service **25** it offers.

[0082] In similar ways that extra methods can be added to a given web service **25**, new virtual services can be composed of methods from various other web services **25** using the infrastructure **501**. By creating a complete API contract **753** for a virtual service that does not really exist, the infrastructure **501** can then route the client calls **701** made to said virtual service methods to the real web services **25** that implement the WS methods **702**. These virtual services could be composed by a system administrator through an application that interfaces with the infrastructure's databases.

[0083] Again, through modification of the API contract **753** whenever it is requested, the gateway module **500** is able to make it appear that a web service **25** resides at any virtual location. By ensuring that the WSDL processor of the gateway module **500** intercepts references to the web service **25** at this virtual location, the gateway module **500** can then route the call to one of the physical locations where it actually exists. When a web service **25** changes its physical location, it is just a matter of updating the data table in the web service registry repository **530** that indicates its location to the gateway module **500**. The original entry in the web service registry repository **530** may be created through a gateway module system administration application when a web service **25** is registered with the gateway module **500**. Clients need not even be aware that the web service **25** has moved.

[0084] **FIG. 9** shows another example of a gateway module **900** in accordance with an embodiment of the present invention. The gateway module **900** includes a client application interface unit **310**, a communication processor **311**, a web services interface unit request dispatcher, **312**, a web services registry repository **530**, a metering module **950**, a web method call processor **960**, a web service API contract processor **965**, a billing module **970**, and a login services module **980**. The login services module comprises an authentication module **520**, an authorization module **525**, an authentication identifier (ID) provider **940**, and an authentication ID validator **945**. Components may be added to or removed from the gateway module **900**.

[0085] The client application interface unit **310**, communication processor **311**, web services interface unit **312**, authentication module **520**, authorization module **525**, and web services registry repository **530** may be similar to those described above. The metering module **950** keeps track of the usage of web service client call **701** methods for the specific client application **15**, including the number of client calls **701** made and amount of server resource consumed. The web method call processor **960** comprises code to perform the modifications to the method calls **701**, **702** and responses **703**, **704** described above. The web method call processor **960** may be a SOAP processor, or any suitable processor for other web method protocols. The API contract processor **965** comprises code to perform the modifications to the API method calls **701**, **702** and WSDL **703**, **704** described above. The API contract processor **965** may be a WSDL processor, or any suitable processor for other web service API. The billing unit **970** comprises code to bill client applications **15** for the transparent use of web services **25**. The login services module **980** comprises code to administer and service a login request received from a client application **15**, and to administer authorization and authentication of a client application **15**. The login request may be passed directly to the login services module **980** from the client application interface unit **310**. Alternatively, the login request may be first sent to the communication processor **311** to be sent to the login services module. The authentication ID provider **940** may comprise code which assigns one or more authentication IDs to a client application **15** when the client application **15** logs into a web service **25**. The authentication ID validator **945** may comprise code to validate the authentication ID. These components will be further described below.

[0086] **FIG. 10** shows another example of a method for managing functionality for one or more web services **25** (**1000**) in accordance with an embodiment of the present invention. The method (**1000**) begins with listening for communications between client applications **15** and web services **25** (**1001**). The communications may be client applications **15** attempting to log into web services **25**, client applications **15** sending web service method calls **701** or API